A Content-Oriented Data Model for Semistructured Data

Tomáš Novotný

Czech Technical University, Faculty of Electrical Engineering,
Technická 2, 166 27 Prague 6, Czech Republic
novott2@fel.cvut.cz

Abstract. There are several data models that are capable of handling semistructured data. The best known are OEM, XML DOM, RDF, and the ECMAScript object model. All these models have different purpose. OEM was used by systems for integration of heterogeneous data sources. XML DOM is specified as a programming interface to manipulate XML documents used as a unified medium for data exchange. RDF provides primarily a data model for sharing metadata. The ECMAScript object model is widely used to manipulate data in web applications. However, none of these models is intended to be used directly in an interactive way. This paper presents the CO (Content-Oriented) data model, which is designed for users to browse, annotate, and relate pieces of information. It can provide change notifications hence it can be directly used in interactive applications without building an extra object model.

Keywords: data entity, data aspect, informed list, informed property

1 Introduction

Traditional mainstream databases save data according a schema that describes the context of the data. They usually need to specify the schema in advance. This requirement is for certain data very difficult or even impossible to fulfill. It is a case of irregular data or data that structure changes over time [1].

Another option is to store data together with a description of its meaning. Such data do not have to be structured according a schema and therefore they are referred to as semistructured data [2]. Data models for semistructured data can be easily used by knowledge management systems or systems for integration of heterogeneous information sources [3].

The existing well-known data models for semistructured data are not intended to be directly used in interactive applications. Programming interfaces around these models are usually based on elaborate query languages that simplify locating and extracting of particular information from these models. However, the more advanced queries can be answered, the more complex systems have to be built to detect changes. Therefore, there is usually no unified support for events that notifies user interface controls when

J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2007, pp. 55–66, ISBN 80-7378-002-X.

something has changed hence such data models cannot be directly used in interactive applications.

This paper describes the CO data model that unlike other data models for semistructured data is intended for interactive applications, especially applications that allow users to browse, annotate, and relate pieces of information. It can be also used for visual manipulation with data stored in other data models or exchange formats for semistructured data like XML or JSON. It is designed to be very simple in order to keep implementation simple.

The CO data model has a modular architecture. A core provides very limited functionality enabling basic manipulation with data. The core itself is not intended for interactive applications as the change notifications are not supported. Nevertheless, it may be used by scripts that are using the same data model. Other features are provided by external modules called extensions. This extensible architecture allows developers to choose particular configuration that will support just required features without unwanted ones. It also enables having various implementations of the same feature in order to enable direct data binding for various user interface toolkits.

Other data models for semistructured data and systems for that they were built are described in the second chapter. A design the CO data model and an implementation of a framework for the CO data model are presented in the third and fourth chapters. Conclusions and a future work are given in the last chapter.

2 Related Work

With the development and growth of the Web [4] in the last decade of the last century there arose a need to extract and integrate data that are available on the Web [5]. At the same time, there was a research project called TSIMMIS whose aim was to create a system for assisted integration of data either structured (e.g. database records), semistructured (e.g. web pages), or unstructured (e.g. plain files) from heterogeneous data sources [6].

2.1 TSIMMIS

TSIMMIS was a joint project between Stanford and the IBM Almaden Research Center [3]. It had mediator architecture [7] [8] and it basically consisted of four types of core components: translators, mediators, constraint managers, and classifier/extractors. Translators were template-based wrappers [9] that converted data from various sources into a common information model. Mediators were information routers that forward queries to particular translators and merge the results [10]. Constraint managers [11] ensured that the integrated data are consistent. Classifier/extractors automatically classified and extracted key properties of resources from the unstructured data sources. They were based on the Rufus system developed by the Almaden Research Center [12].

Information between these components was exchanged in a self-describing object model called OEM (Object Exchange Model) [13]. OEM allowed the storing of nested objects. Each object was represented by a structure with four fields: label, type,

value, and oid. The label was a string tag that describes what the object represents. The type was a data type of the object's value. The value stored the actual data. The oid was a unique identifier of the object.

TSIMMIS was not a fully automated system, but rather a tool to assist humans. It provided a graphical user interface component called MOBIE (MOsaic Based Information Explorer) [14] that allowed users to browse OEM objects and specify queries in an SQL-like language called OEM-QL [13].

2.2 LORE

LORE (Lightweight Object REpository) was a database management system designed to manage semistructured data [1]. It was built on top of the O^2 object database [15]. Semistructured data were stored in a modified OEM that was presented in the TSIMMIS project. That version of OEM could be represented as a labeled directed graph where the vertices were objects. There were two types of objects: complex objects and atomic objects. Whereas the complex objects might have outgoing edges to other objects, the atomic objects had no outgoing edges but they contained a value from one of the atomic types (e.g. integer or string).

LORE provided a query language called LOREL [16]. LOREL was defined as an extension to OQL (an SQL-like query language for the ODMG model) [17]. LOREL was also later used as a query language for TSIMMIS [8].

LORE introduced a DataGuide [18]. The DataGuide was a structure summary of the OEM model that was automatically maintained. DataGuides allowed users to browse the OEM model and formulate queries. They were also used by the system to store statistics and to optimize the queries. The DataGuide itself was an OEM object so it could be stored in and managed by an OEM DBMS.

Another feature of LORE was an external data manager [19]. The external data manager allowed users to integrate data from external data sources. The external data were represented by an external object. The external object was stored in the OEM database and it contained both a specification on how to fetch the external data and a cached version of the external data. The wrappers for the external data were reused from the TSIMMIS project.

After the emergence of XML, developers of LORE found that data models of XML and OEM were similar. So they decided to modify LORE to serve as a data management system for XML [20]. The modifications to LORE also required changes to the data model [21].

2.3 XML

XML is a markup language designed for data exchange [22]. The data are represented as documents. The XML document is a tree-like structure built from nested tagged elements. Each element can contain data stored as attributes/value pairs or as a plain text. XML also provides a mechanism to create links between elements.

There are several ways to extract information from XML documents. None of them can be considered as universal but each is convenient for a particular purpose. One of

the best known is SAX. SAX (Simple API for XML) provides a read-only and forward-only event-driven interface [23]. Another read-only interface is XmlReader (not to be confused with XMLReader, the Java interface from the SAX2 library). In contrast to SAX, XmlReader allows developers to navigate through XML on-demand in the way that is sometimes referred to as a pull model [24]. More sophisticated navigation provides XPathNavigator that enables cursor-like navigation in a XML document powered by XPath expressions [25].

There is also a standardized virtual data model for XML called DOM (Document Object Model) that is specified as a programming interface and it is developed to manipulate a memory representation of XML documents [26].

2.4 RDF

RDF (Resource Description Framework) is a set of specifications [27] that was created to provide a unified way to share metadata and it can be also used to represent data [28]. Although RDF is more complex than previous models, the data model of RDF can be represented by a collection of triples [29]. Each triple consists of a subject, a predicate, and an object. Predicates are also called properties. Subjects may be URIs or blanks. Predicates are URIs. Objects can be URIs, literals, or blanks. URIs may be used both as references to existing resources and as a global identifiers. Literals represent values. They may be plain or typed. Blanks are local identifiers.

There are various languages that extend capabilities of RDF. The best known are RDF Schema and OWL. RDF Schema allows users to describe classes and properties [30]. OWL (Web Ontology Language) enables data description using ontologies [31].

2.5 ECMAScript object model

The ECMAScript object model is widely used by web applications to manipulate data [32]. Structure of the ECMAScript is similar to LORE's version of OEM. The difference between LORE's OEM and the ECMAScript object model is that objects in the ECMAScript object model have no identifiers and each object can have only a single property with the same name. Multiple values can be represented by a special build-in type of object – an array. The textual representation of this model is sometimes referred to as JSON [33]. JSON has similar purpose as XML. There is also a standard 'ECMAScript for XML' that adds native XML support to ECMAScript [34].

2.6 iDM

iDM is an advanced data model that is designed to represent heterogeneous data [35]. iDM is now being developed as a part of a personal information management system iMeMex [36].

3 Design

As mentioned in the introductory chapter, this paper describes the CO data model that is designed for interactive applications. This implies that except of developers, the design should also take account of users. There are three elemental requirements: The first is that the data model should be simple because the simplest it would be, the easiest the implementation of a framework for the data model would be. The next is that the data model has to be extensible so the developers can use only that parts what they really need. The last requirement is that it should have similar concepts as existing systems for information handling that are widely used by users, so that users do not need to spend time for an extra training.

The most famous and widely used system that is used by users to access information is the Web. The information on the Web is usually stored in a document called a web page. There are two basic ways to access a particular web page: users can either enter the address of the web page or navigate to a particular web page from another web page. Advanced methods include searching by entering a query into a search engine or navigating through tags. Tags are usually keywords that are attached to a piece of information. In fact, navigation through tags is mostly a particular form of simple page to page navigation, as tags are usually located within the page.

The data model should offer thus similar ways of accessing information as the Web: The data model should allow users to navigate through the data and access the data directly from an address, enable developers to easily create tagging systems, and provide programming interfaces so that specialized search engines can be built or adapted to process search queries.

3.1 Data model

The architecture of the CO data model results from its requirements. The data model consists of items called entities. The entity is a wrapper for data. The data contained in the entity will be referred to as content. In addition to the content, each entity has a type and optionally has references to other entities.

The content of an entity can be anything. Entities can store numbers, text documents, multimedia files, an array of objects or other entities, or any other pieces of information. Entities do not need to contain data directly, but they can contain special objects that refer to external data stored in local files, databases, or even remote web sites.

The entities' types have a similar purpose as the labels in TSIMMIS's version of OEM or the tags in XML, with the difference that labels in OEM and tags in XML are strings but entities' types are represented by another or, in special cases, the same entity. This architecture allows saving common metadata to the entity that represents the type. Metadata can contain both information for humans, such as documentation, and information for machines, for instance specifications of constraints or processing instructions for manipulation of external data that are referred in the content. The fact that the type of entity `A` has is entity `B` can be expressed in RDF by a triple `<entity:A> <rdf:type> <entity:B>`.

Entity reference is consists of a key and a value. It is a simplified version of the properties in RDF. Both the key and value of a reference are entities. In contrast to RDF, the value of a reference cannot be a literal. As with the ECMAScript object model, entity can have at most one reference with the same key. As the reference key is an entity, the entity can store metadata that can provide information such as semantic meaning, usage constraints, or human-readable documentation. Fig. 1 shows the data model of the entity.

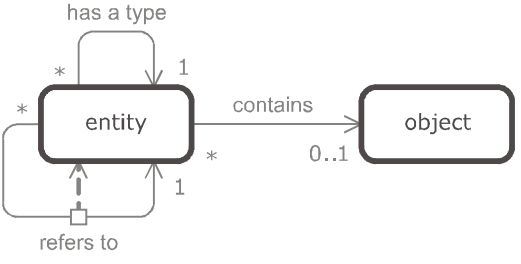

Fig. 1. The data model of the entity.

Entities can be directly accessed by URIs. However, URIs are not mandatory for entities. An entity without a URI can be accessed by searching or by local navigation.

This part described the core part the CO data model. Additional features can be provided as extensions. Following part describes representation of several basic data structures.

3.2 Data structures

Unlike RDF or the ECMAScript object model, the CO data model has no direct support for collections. As collections are the most common data structures, this part sketches how to form collections in the CO data model.

There are basically two ways to build a collection. The first way is to create a special data type that represents a collection, so the collection will be stored in the content of an entity. This method allows the storing of any type of collection. It is especially useful for vectors or matrices and it can be likened to containers in RDF or arrays in the ECMAScript object model.

The second method uses references to group items in a collection. There are two types of collection that can be formed. The first type includes collections whose entity directly refers to multiple items. Basic examples of such collections are simple or hierarchical dictionaries.

The other type of collection points directly to just one item. One of the simplest examples is a singly linked list. The singly linked list is usually formed by adding to each item a reference that points to the following item. There can be also a special object that represents a whole collection. This object has a reference to the first item in the collection. In this scenario, items of a collection do not know in what collection they are contained without an additional reference and they can be usually contained in a single collection.

The CO data model enables the creation of specialized singly linked lists where each item can be contained in multiple collections and each item knows what collections it is contained in without additional references. Such linked lists will be referred to as informed lists. The architecture of an informed list is as follows.

The entity that represents a collection remains unchanged – it has a reference to the first item. Items also contain references to the next item. However these references are not identified by a general next entity but by the entity of the collection. As each collection is represented by a different entity, items have references with different keys and hence they can be contained in multiple collections. Items also know the collections where they are included because this information is stored in the keys of the references to the next items. Fig. 2 shows an example of an informed list.

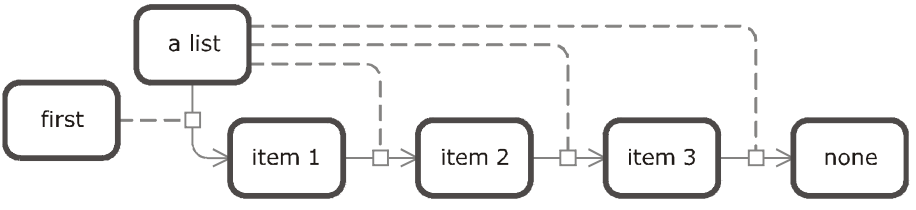

Fig. 2. Example of an informed list. The list is represented by entity “a list”. It contains three items: “item 1”, “item 2”, and “item 3”. Entity “first” represents a known entity that is used as key that refers to the first item and “none” is another known entity needed to store reference to the owning list in the last item. Entities are represented by boxes, references are rendered as solid arrows, and keys of the references are pointed by dashed lines.

One possible application for informed lists is a tagging system where each tag is represented by a collection of entities that are tagged by this tag. A tagging system can list all tags for a resource in a constant time without any additional indexes. It can also immediately retrieve an additional resource for each tag. This feature can be useful to provide a simple and fast tag-based ‘see also’.

Several informed lists can form a multilevel hierarchical structure where descendants know their ancestors. One of the possible applications is a system for hierarchical categorization.

The idea of informed lists can be also used to construct properties that can have multiple values and that know what objects refer to them. Such properties will be called informed properties. This construct can be conceived as a fusion of informed lists. Each list contains values for a particular property. And the key of the reference to the first value in each list is replaced by a key that will identify the property. An example of informed property is in Fig. 3.

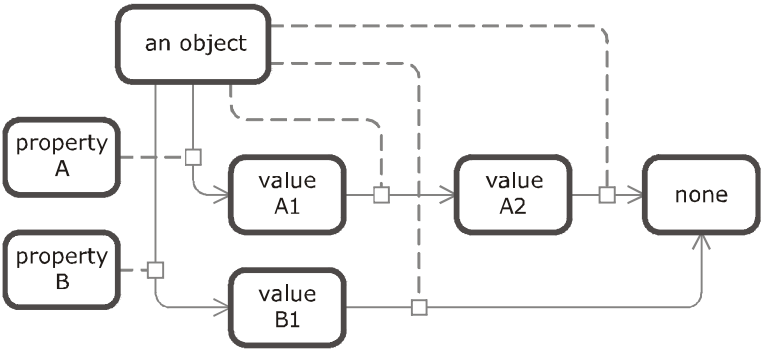

Fig. 3. Example of informed properties. Entity “an object” has two informed properties: “property A” and “property B”. Property “property A” has two values “value A1” and “value A2” and property “property B” with one value “value B1”.

4 Implementation

This chapter is divided into two parts. The first part outlines the general architecture of a framework that implements the CO data model. The second part describes a prototypical implementation to evaluate the CO data model.

4.1 Architecture

All frameworks for the CO data model should implement the core of the data model. The core was described in the Architecture part of the Design chapter. A core framework should provide programming interfaces to access entities and to get and set a type, references, and content. There are no methods to create or delete entities because each URI refers to an existing entity. The consequence is that there is an infinite number of entities. However, only a finite number of entities contain non-default information hence the framework should store only the entities with non-default information. The entities with default information are called implicit entities and the other entities are called explicit entities. The implicit entities have null content, no references, and a default type.

The minimal framework may implement only a volatile data model. Such a framework can be useful, for instance, in applications that use existing web services to store data.

Other features are optional and they are not part of the core. They are provided by extensions. There are several types of extensions. One type of the extension is a data aspect. The data aspect is a component that simplifies manipulation with a complex data structure, such as the informed list. The data aspect for the informed list can provide methods to add, remove, or search items. It can enhance the performance by cached backward links or a cached index array for immediate random access.

Another type of extension is a data store provider. The data store provider allows loading and saving data to data storages either local (e.g. files, databases) or remote

(e.g. web services). They can also serve as data adapters between the CO data model and internal object models of other applications.

Moreover, interactive applications require knowing when something has changed. This implies enhanced implementation of the core that raises an event if a change occurs. The system for event notification may be developed for a certain user interface toolkit so the application developers can directly bind controls to the data model.

4.2 Prototype

Currently, there is a prototype of a framework for the CO data model. It is written in C# [37] and runs on the .NET 3.0 Framework [38] [39]. The data model is implemented as .NET Component Model [40]. It was specially developed for the Windows Presentation Foundation [41]. It implements a data aspect for the informed list that caches backward links and an index array to provide faster random access. The data aspect also provides collection-change notifications so it can be bound to standard controls that display collections.

The framework supports simple transactions that delays commit and provides rollback of changed data. There are four data store providers: file system provider, SQL provider, Lucene.Net [42] provider, and Berkeley DB [43] provider.

4.3 Evaluation

The architecture of the data model allows simple implementation of the data model as .NET Component Model with change notifications. Therefore, application developers can directly bind data to user interface controls. Two-way data binding [44] let them develop applications where users can interactively manipulate data while the developers do not have to build an extra object model and write a glue code to transfer data between data stores and user interface controls.

5 Conclusion

The extensible architecture of the CO data model has some advantages and drawbacks. The main advantage is that complex features can be later replaced with a better implementation. However, it can result in several implementations of the same feature and developers have to choose one that best fits their needs, so they may spend extra time analyzing various implementations. This can be avoided, of course, by maintaining a list of various implementations with comparisons and use cases.

As is mentioned in the introduction, in contrast to OEM, XML, and RDF, the CO data model is intended to be directly manipulated by users through user interface controls. Therefore, before a development of other extensions is undertaken, this model must prove that it is really useful and that future work on this model will not be a waste of time. Such proof may be done by building small applications that will help users to organize information from various aspects, such as a system to store personal thoughts or to manage personal data from other applications.

If the data model will be successful, it can be later enhanced by other extensions demanded by users. Some possible extensions might be a change log to enable full undo functionality or allowing users to share their information on a peer-to-peer network and letting them specify particular access rights for particular users on a particular set of their information.

Acknowledgments. I would like to express many thanks to Christoph Quix, who has taught me to write a paper and with whom I was consulting the content of this paper, and Roger Hughes, who made a deep language correction of the draft.

References

1. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3), pages 54-66, 1997.
2. P. Buneman. Semistructured data. *In Proc. of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 117-121, 1997.
3. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. *In Proc. of IPSJ Conference*, pages 7-18, 1994.
4. T. Berners-Lee, R. Cailliau, J. Groff. The World-Wide Web. *Computer Networks and ISDN Systemts*, 25, pages 454-459, 1992.
5. O. Etzioni. The World-Wide Web: quagmire or gold mine? *Communications of the ACM*. Volume 39, Issue 11, 1996.
6. J. Hammer, J. McHugh, H. Garcia-Molina, Semistructured Data: The TSIMMIS Experience, *In Proc. ADBIS'97*, St. Petersburg, Russia, 1997.
7. G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer 25*, 1992.
8. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajamaran, Y. Sagiv, J. Ullman, V. Vassalos and J. Widom, The TSIMMIS approach to Mediation: Data Models and Languages, *Journal of Intelligent Information Systems*, 8(2) pages 117-132, 1997.
9. J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, and V. Vassalos. Template-based wrappers in the TSIMMIS system. *In Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD '97)*, pages 532-535, 1997.
10. Y. Papakonstantinou, S. Abiteboul, and H. GarciaMolina. Object fusion in mediator systems. *In Proc 22nd. VLDB conference*, 1996.
11. S. Chawathe, H. Garcia-Molina, J. Widom. Constraint Management in Loosely Coupled Distributed Databases. Technical report, Computer Science Department, Stanford University, 1993.
12. K. Shoens, A. Luniewski, P. Schwarz, J. Stamos, and J. Thomas. The RUFUS System: Information Organization for Semi-Structured Data. *Proceedings of the International Conference on Very Large Databases*, pages 97-107, Dublin, Ireland, 1993.
13. Y. Papakonstantinou, H. Garcia-Molina, J. Widom. Object Exchange Across Heterogeneous Information Sources. *In Proc. of 11th International Conference on Data Engineering (ICDE'95)*, pages 251-260, Taiwan, 1995.
14. J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom, Information translation, mediation, and mosaic-based browsing in the TSIMMIS system. *In Proc. of the ACM SIGMOD Internation Conference on Management of Data*, 1995.

15. O. Deux. The O2 system. *Commun. ACM 34*, num. 10, pages 34-48 1991.
16. S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. Weiner. The Lorel Query Language for Semistructured Data. *Journal of Digital Libraries*, 1(1), pages 68-88, 1997.
17. R. Cattell, T. Atwood. *The Object database standard, ODMG-93*. Morgan Kaufmann Publishers Inc. 1994, ISBN 978-1558603028.
18. R. Goldman, J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. *In Proc. VLDB*, pages 436-445, 1997.
19. J. McHugh, J. Widom. Integrating Dynamically-Fetched External Information into a DBMS for Semistructured Data. *SIGMOD Record*, 26(4), pages 24-31, 1997.
20. J. Widom. Data management for XML: Research directions. *IEEE Data Engineering Bulletin*, 22(3), pages 44-52, 1999.
21. R. Goldman, J. McHugh, J. Widom. From semistructured data to XML: Migrating the Lore data model and query language. *In Proc. of the WebDB workshop*, 1999.
22. T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition), *W3C Recommendation*, 16th August 2006.
23. Simple API for XML. http://www.saxproject.org/
24. A. Skonnard. XML in .NET: .NET Framework XML Classes and C# Offer Simple, Scalable Data Manipulation. *In MSDN Magazine*, January 2001.
25. D. Esposito. Manipulate XML Data Easily with the XPath and XSLT APIs in the .NET, *In MSDN Magazine*, July 2003.
26. W3C. Document Object Model. *W3C Recommendation.*
27. F. Manola, E. Miller. RDF Primer. W3C Recommendation 10 February 2004.
28. S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. Klein, J. Broekstra, M. Erdmann, I. Horrocks. The Semantic Web: The Roles of XML and RDF. *IEEE Internet Computing*, 4 (5), pages 63-74, 2000.
29. G. Klyne, J. Carroll. RDF Concepts and Abstract Syntax. *W3C Recommendation*, 10th February 2004.
30. D. Brickley, R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendation*, 10th February 2004.
31. D. McGuinness, F. van Harmelen. OWL Web Ontology Language. *W3C Recommendation*, 10th February 2004.
32. ECMA. ECMAScript Language Specification. *Standard ECMA-262 3rd Edition*, 1999.
33. D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). *Request for Comments: 4627*, July 2006.
34. ECMA. ECMAScript for XML (E4X) Specification. *Standard ECMA-357 2nd Edition*, 2005.
35. J. Dittrich, M. Salles. iDM: a unified and versatile data model for personal dataspace management. *In Proc. VLDB 2006*, 32, pages 367-378, 2006.
36. J. Dittrich, L. Blunschi, M. Färber, O. Girard, S. Karakashian, M. Salles. From Personal Desktops to Personal Dataspaces: A Report on Building the iMeMex Personal Dataspace Management System. *In BTW 2007*, 2007.
37. ECMA. C# Language Specification. *Standard ECMA-334 4th Edition*, 2006.
38. .NET Framework 3.0 Technologies, *MSDN* http://msdn2.microsoft.com/en-us/netframework/aa663323.aspx
39. .NET Framework 3.0 Community http://www.netfx3.com/
40. System.ComponentModel Namespace, *.NET Framework Class Library, MSDN* http://msdn2.microsoft.com/en-us/system.componentmodel(vs.80).aspx

41. Windows Presentation Foundation, *MSDN Library*
http://msdn2.microsoft.com/en-us/library/ms754130.aspx
42. Lucene.Net
http://www.dotlucene.net/
43. Oracle Berkeley DB Product Family
http://www.oracle.com/database/berkeley-db/
44. Data Binding Overview, *WPF, MSDN Library*
http://msdn2.microsoft.com/en-us/library/ms752347.aspx